Reference-Based Indexing of Sequence Databases[*]

Jayendra Venkateswaran   Deepak Lachwani   Tamer Kahveci   Christopher Jermaine
CISE Department,University of Florida,Gainesville,FL 32611
{jgvenkat, da1, tamer, cjermain}@cise.ufl.edu

ABSTRACT

We consider the problem of similarity search in a very large sequence database with edit distance as the similarity measure. Given limited main memory, our goal is to develop a reference-based index that reduces the number of costly edit distance computations in order to answer a query. The idea in reference-based indexing is to select a small set of reference sequences that serve as a surrogate for the other sequences in the database. We consider two novel strategies for selecting references as well as a new strategy for assigning references to database sequences. Our experimental results show that our selection and assignment methods far outperform competitive methods. For example, our methods prune up to 20 times as many sequences as the Omni method, and as many as 30 times as many sequences as frequency vectors. Our methods also scale nicely for databases containing many and/or very long sequences.

1. INTRODUCTION

Many applications require searching a database of sequences to find similarities to a given query sequence. Genomics, proteomics and dictionary search are just a few examples. Given a database of sequences S, a query sequence q, and a threshold ϵ, similarity search finds the set of all sequences whose distance to q is less than ϵ. Often, edit distance is used to measure the similarity. The edit distance between two sequences is the minimum cost transformation of one to the other by a series of edit operations (insert, delete, modify) on individual characters. The dynamic programming solution to find the edit distance between two sequences runs in $O(n^2)$ time and space [26], where n is the average length of a sequence. The space and time complexity for the bounded version of this problem is $O(n\epsilon)$ [2, 3, 24, 31]. In many applications, $\epsilon = O(n)$, thus making the complexity of the bounded version $O(n^2)$ [15, 19]. For very large databases or those with long sequences, applying dynamic programming for all (query,sequence) pairs is infeasible in terms of the computation time.

*This work is partially supported by the National Science Foundation under Grant No. 0347408.

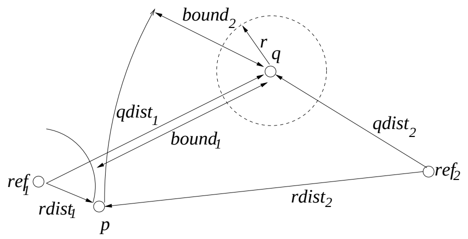


Figure 1: Reference-based indexing: ref_1 and ref_2 are references. The query region is given by its center q and radius r. $qdist_1$ and $qdist_2$ are query-to-reference distances. $rdist_1$ and $rdist_2$ are distances from the reference to the data p. $bound_1$ and $bound_2$ are the bounds obtained using references.

In this paper, we consider the problem of reducing the number of sequence comparisons needed to obtain the results of similarity searches using the method generally referred to as *reference-based indexing* [11, 17, 36]. This is done by selecting a small fraction of sequences referred to as the set of *reference sequences*. The distances between references and database sequences are pre-computed. Given a query, the search algorithm finds the distance from each of the reference sequences to the query sequence. Without any further sequence comparisons, sequences that are too close to or too far away from a reference are removed from the candidate set based upon those distances with the help of the triangle inequality.

Figure 1 illustrates reference-based indexing in a hypothetical two-dimensional space. Here, the sequences are represented by points. The distance between two points in this space corresponds to the edit distances between the two sequences (e.g., $rdist_1$ between the points ref_1 and p corresponds to the edit distance between the sequences represented by them). The edit distance between the data sequence p and reference sequences ref_1 and ref_2 are precomputed. Let $rdist_1$ and $rdist_2$ be these distances, respectively. Given a query q with range r, reference-to-query distances $qdist_1$ and $qdist_2$ are computed. A lower bound for the edit distance between sequences q and p with reference ref_1 is computed as $bound_1 = |qdist_1 - rdist_1|$ using the triangle inequality. Note that edit distance for global alignment is a metric [19]. Similarly, $bound_2$ gives a lower bound for the edit distance between q and p with reference ref_2. Since $bound_1 > r$ with ref_1 as the reference, sequence p can be pruned from the candidate set of q.

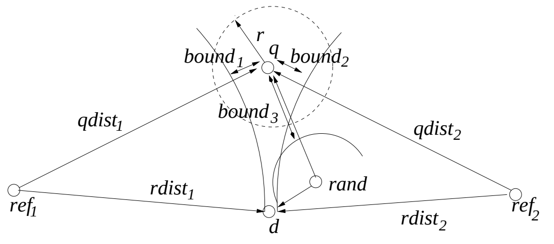


Figure 2: The Omni method: ref_1 and ref_2 are Omni references. The query region is given by its center q and radius r. $qdist_1$ and $qdist_2$ are query-to-reference distances. $rdist_1$ and $rdist_2$ are distances from the reference to data p. $bound_1$ and $bound_2$ are the bounds obtained using the Omni references. $bound_3$ is the bound obtained using a random point $rand$ inside the hull.

Recently, several approaches have been developed for selecting references for reference-based indexing [11, 17]. But these methods perform poorly under certain conditions. For example, the Omni method proposed by Filho et al. [11] selects references near the convex hull of the database, far away from each other. However, this strategy may achieve poor pruning rates. Multiple, redundant references are available to prune the sequences near the hull but no references are available to prune sequences that are far from the hull. Figures 1 and 2 illustrate this problem. In Figure 1, ref_1 and ref_2 are the references from the Omni method. Sequence p is close to ref_1 and far away from ref_2. For query q, p can be pruned by both ref_1 and ref_2, representing a wasted sequence. On the other hand, sequences inside the hull will not be pruned at all, as illustrated in Figure 2. Here, the bounds obtained for the query q and sequence d with two Omni references do not remove d from the candidate set. On the other hand, the reference $rand$ in this figure can prune d. As we will argue in this paper, rather than selecting references near the hull, it is better to choose references so that each prunes a different subset of the database.

Our Contributions: In this paper, we tackle the two primary problems that must be addressed in reference-based indexing. The first one is the selection of references. We select references so that they represent *all* parts of the database. We propose two novel strategies for the selection of references. They are:

Maximum Variance: This approach follows from two observations related to the triangle inequality property of the edit distance: i) If a query q is close to reference v, we can prune sequences far away from v and add sequences close to q to result set, and ii) If q is far from v, we can prune sequences close to v. These observations imply that the spread of the database sequences around a reference sequence is a good measure for a reference. Thus, our method selects references among sequences with higher variance of distances.

Maximum Pruning: Our second approach is based upon a combinatorial calculation that specifically selects references to maximize the pruning of sequences. It is based on the following two rules: i) Select references based on the number of sequences it can prune, and ii) Sequences not pruned by one reference should be pruned by at least one of the other references. Since the complexity of the method increases with database size, we use sampling techniques to speed the process.

The second problem we consider in the paper is the mapping of references to database sequences. In our method, the number of references in the reference set is larger than in other methods. In order to keep the number of references assigned to each sequence manageable, only the best references for each given database sequence are used to index that sequence. Thus, each database sequence may have a *different* set of reference sequences assigned to it.

Paper Organization: The rest of this paper is organized as follows. Section 2 describes our similarity measure, how to obtain bounds, and gives a cost analysis. Section 3 presents background on the current index structures for database search. Section 4 describes our first method for selecting references, *Maximum Variance*. Section 5 presents our second method for selecting references, *Maximum Pruning*. Our strategy of assigning references to the database sequences is given in Section 6. Given the references and their assignments to sequences, Section 7 describes our search algorithm which computes the set of candidate sequences from the database without requiring actual distance computations. Section 8 presents experiments showing the performance of our methods and a comparison with several of the existing access methods under different parameter settings. Section 9 gives the conclusion of the paper.

2. BACKGROUND

Edit Distance: The *edit distance* between two sequences P and Q is the number of edit operations (insert, delete, transform) required to transform P into Q. We will use $ED(P,Q)$ to denote the function returning the edit distance between P and Q. ED is a metric distance, i.e. given two sequences P and Q, it has three properties: a) symmetry ($ED(P,Q) = ED(Q,P)$), b) it is always non-negative ($ED(P,Q) \geq 0$), and c) it satisfies the triangle inequality ($ED(P,X) \leq ED(P,Q) + ED(Q,X)$ for any sequence X).

Reference-Based Indexing: Given a database of sequences S, a query sequence q and a range ϵ, the goal is to find all database sequences within distance ϵ from q. Let $V = \{v_1, \cdots, v_m\}$ denote the set of reference sequences, where $v_i \in S$ and $|V| = m$. In reference-based indexing, we pre-compute distances to database sequences from the references given by the set $\{ED(s_i, v_j) | (\forall s_i \in S) \wedge (\forall v_j \in V)\}$. This is a one-time cost for the database.

To use the references, a search algorithm first computes the distances to the query sequence q from the reference sequences. For each sequence s_i, a lower bound (LB) and an upper bound (UB) for $ED(q, s_i)$ are given by:

$$LB = \max(\bigvee_{v_j \in V} |ED(q, v_j) - ED(v_j, s)|$$
$$UB = \min(\bigvee_{v_j \in V} |ED(q, v_j) + ED(v_j, s)|$$

LB and UB are used in reference-based indexing as follows. For each (q, s_i) pair:

- If $r < LB$, add s_i to the *pruned set.*
- If $r > UB$, add s_i to the *result set.*
- If $LB \leq r \leq UB$, add s_i to the *candidate set.*

After this pruning, sequences in the candidate set are compared with q using dynamic programming to complete the query.

Cost Analysis: The factors that determine the cost of strategies used for selection and assignment of references to database sequences are *memory* and *computation time.*

Memory: Let the available main memory be B bytes, the number of references assigned be k, and let V be the reference set. We assume four bytes are used to store an integer and a sequence uses on average z bytes of storage. Thus, zk bytes are needed to store a reference. For each sequence $s \in S$ and its reference $v_i \in V$, the sequence-reference mapping is of the from $[i, ED(s, v_i)]$ using two integers. Thus, $8kN$ bytes are used to store the pre-computed edit distances for N sequences. The number of references k can be obtained by comparing the available memory with the memory needed for storage: $B = 8kN + zk$.

Computational Complexity: Let Q be the given query set, t be the time taken for one sequence comparison and c_{avg} be the average number of sequences in the candidate set for each query. First, each query sequence is compared with each of the k references. This takes $tk|Q|$ time. Then for each query, the c_{avg} sequences have to be compared to filter out the false positives. This takes $tc_{avg}|Q|$ time. The total time taken is given by $t(k + c_{avg})|Q|$.

3. RELATED WORK

Searching sequence databases involves retrieving database sequences that are similar to a given query. A number of index structures have been developed to reduce the cost of such searches. They can be classified under three categories: k-gram indexing, suffix trees, and vector space indexing.

A k-gram is a sequence of length k, where k is a positive integer [13]. k-gram based methods look for the shortest subsequences that match exactly; these sequences are then extended to find longer alignments with mismatches and inserts/deletes. k-grams are usually indexed using hash tables. Two of the most well known genome search tools that use hash tables are FASTA [27] and BLAST [1]. The performance of these tools deteriorates quickly as the size of the database increases.

Suffix trees were first proposed by Weiner [35] under the name *position tree*. Later, efficient suffix tree construction methods [22, 32] and variations [10, 16, 20, 21] were developed. However, there are two significant problems with the suffix-tree approach: (1) suffix trees manage mismatches inefficiently, and (2) suffix trees are notorious for their excessive memory usage [25]. The size of the suffix tree varies between 10 to 37 bytes per letter [9, 16, 20, 23].

A number of index structures have been developed to function in vector space, such as SST [12] and the frequency vectors [19, 18]. The frequency vector of a sequence stores the number of letters of each type in that sequence. This method computes a lower bound to the distance between two sequences using the frequency vectors corresponding to the two sequences. It uses this lower bound to eliminate unpromising sequences. However, as the query range increases frequency vectors perform poorly.

Reference-based indexing [7, 11, 17, 36] can be considered as a variation of vector space indexing. One method, the VP-tree [36], partitions the data space into spherical cuts by selecting random reference points from the data. A second method, the MVP-Tree [6] (a variation of VP-Tree) uses more than one vantage point at each level. [28] uses the VP-Tree to index measures that are *almost metric.* iDistance [17] proposes two principal approaches for selecting reference points. For normally distributed data, the d-dimensional space is partitioned into $2d$ pyramids. Each partition is allocated a reference point, which can be the centroid of the pyramid base or an external point. For correlated data, the most frequently occurring values in each dimension are used to select reference points. Filho et al. [11] proposed one of the recent reference-based indexing methods, called *Omni.* In Omni, reference sequences are selected from the convex hull of the dataset. This is done by selecting sequences that are far away from each other. A similar approach for selecting references in content-based image retrieval has been proposed in [34].

The M-Tree [8] is a height-balanced tree with the data objects in its leaf nodes. A variation of the M-Tree, the Slim-Tree [30], reduces the amount of overlap between the tree nodes, referred as *Slim Down.* These tree-based structures are height balanced and attempt to reduce the height of the tree at the expense of flexibility in reducing the overlap between the nodes. This constraint was relaxed in the DBM-Tree [33] by reducing the overlap between nodes in high-density regions, resulting in an unbalanced tree. The DF-Tree [29] selects a global set of representatives in a manner similar to Omni in order to prune candidate sequences when answering queries.

4. MAXIMUM VARIANCE

In this section, we describe the first of our two heuristic strategies for selecting the reference sequences from the database. As described in Section 1, an important issue overlooked in previous work is that the performance of reference-based indexing can be improved by selecting references that have a significant number of sequences close to and far from them. The variance of the distribution of distances of a reference to other sequences is a good indicator of the spread of sequences in the database around that reference, and we make use of it in our first heuristic.

Our Maximum Variance heuristic assumes that queries follow the same distribution as the database sequences. It selects a reference set that represents the sequence distribution of the database. Each new reference prunes some part of the database not pruned by the current sequences in the reference set.

This suggests an algorithm for choosing reference points. For example, in Figure 3, points in two-dimensional space represent the database sequences. The edit distance between two sequences corresponds to the distance between the points representing them. Sequence e has the highest variance of distances. So we choose e as the first reference. Sequences close to e (sequences f and g) and far from it (sequence b) can be pruned using e as the reference. The sequences not pruned by e (sequences a, c and d) remain in the candidate set. A sequence from the candidate set having the next highest variance, c, is selected as the next reference. Reference c can remove the sequence d close to it and a far from it from the candidate set. This process is repeated until all of the references have been assigned.

Let L denote the length of the longest sequence in S. For a sequence $s_i \in S$, μ_i and σ_i are the the mean and variance of its distances to other sequences in S. We compute a cut-off distance, w, to measure the closeness of two sequences. We say that s_j ($s_j \in S$) is *close to* s_i if $ED(s_i, s_j) < (\mu_i - w)$ and s_j is *far away* from s_i if $ED(s_i, s_j) > (\mu_i + w)$. We

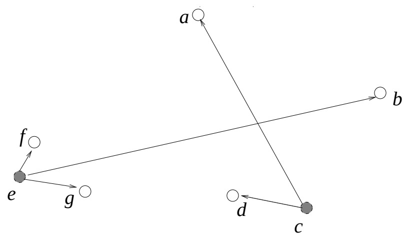


Figure 3: Maximum Variance example: The dataset is given by the sequences $\{a, b, c, d, e, f, g\}$. Sequences are represented by shaded points. e and c are the references selected by Maximum Variance. Sequences f, g are close to e and b is far from e. Sequence d is close to c and sequence a is far from c.

compute w as a fraction of L, given by $w = L.perc$, where $0 < perc < 1$. We do not consider s_j as a potential reference if $ED(s_i, s_j) < (\mu_i - w)$ or $(\mu_i + w) < ED(s_i, s_j)$, $\exists s_i \in V$. A large value for *perc* will include sequences that are *close to* or *far away* from the existing references. This results in sequences being pruned by multiple references. A small value for *perc* can remove promising references, resulting in a small number of references. Experimental results show that $perc = 0.15$ is a good choice.

Figure 4 presents the algorithm in detail. For each sequence $s_i \in S$, in Step 2.a we first select a sample database S', $S' \subset S$ and compute the set of distances $D_i = \{ED(s_i, s_j) \mid \forall s_j \in S'\}$ (Step 2.b). In Step 2.c, we compute the mean μ_i and variance σ_i of the distances in D_i. The distances are then sorted in descending order of their σ_i values (Step 4). Then, we repeatedly do the following until the required number of references is obtained. We select the sequence s_1 with maximum variance as the next reference and add it to V (Step 5.a) and remove the sequences from S that are *close to* or *far away* from the new reference s_1 (Step 5.b). Steps 5.a and 5.b are repeated until we have enough references, i.e. $|V| = m$. At each iteration we select a new reference that is neither close to nor far away from the existing references.

Computational Complexity: Step 2 compares each candidate reference with all sequences in the sample, S'. This requires $O(N|S'|)$ distance computations. Each distance comparison takes $O(L^2)$. Sorting the variances of N sequences in Step 3 takes $O(N \log N)$. Steps 5.a-5.c take $O(mN)$ time in the worst case. Thus the overall time of the algorithm is $O(NL^2|S'| + N \log N + mN)$.

5. MAXIMUM PRUNING

In this section, we consider a second approach for choosing the reference set, which combinatorially tries to compute the best reference set for a given query distribution. Obviously, a purely combinatorial approach would be very expensive. Let S denote the set of database sequences where $|S| = N$. Let Q denote the sample query set. Exhaustively testing all possible combinations of m references from S takes $O(C_m^N \times N \times |Q|)$, where C_m^N is N choose m. To speed up this computation, we propose a greedy solution to this problem which reduces its complexity. We refer to our method as Maximum Pruning. In order to speed the greedy solution even further, we also consider sampling-based optimizations for this algorithm in Section 5.2.

```
/*Input:Sequence database $S$, with |$S$| = $N$.
        Number of references $m$.
        Cutoff percentage $perc$.
        Length of a sequence $L$.
  Output:Set of references $V = \{v_1, v_2, \ldots, v_m\}$*/
1. $V = \{\}$. /* Initialize */
2. For each $s_i \in S$ do
   (a) Select sample set of sequences, $S' \subset S$.
   (b) Compute $D_i = \{ED(s_i, s_j) \mid \forall s_j \in S'\}$.
   (c) Compute mean $\mu_i$ and variance $\sigma_i$ of the distances in $D_i$.
3. $w = L.perc$.
4. Sort the $N$ sequences in descending order of their variances.
5. While $|V| < m$ do
   (a) $V = V \cup s_1$.
   (b) $S = S - \{s_j\}$, $\forall s_j \in S$ with $ED(s_1, s_j) < (\mu_1 - w)$
       or $ED(s_1, s_j) > (\mu_1 + w)$. /* Remove sequences
       close to or far away from the new reference */
6. Return of set of reference sequences, $V$.
```

Figure 4: Maximum Variance method.

5.1 Algorithm

Our method considers each sequence in the database as a candidate reference. The method starts with an initial reference set. It then iteratively refines the reference set. At each iteration, it replaces an existing reference with a new reference if this modification improves pruning with respect to Q. The method stops the process if the reference set cannot be improved.

Figure 5 presents our Maximum Pruning algorithm. The algorithm takes a database of sequences S, a query set Q, and the number of references m as input. It first initializes the reference set V as the top m references obtained using Maximum Variance method (though this is not a requirement, since one can start with a random initial reference set). Every execution of the Do-While loop replaces one existing reference with a better reference. Each iteration of this loop starts by initializing the array G to zero. Each entry $G[i]$ of G shows the amount of pruning gained by including the ith candidate reference in the reference set. We use the term *gain* to denote the amount of improvement in pruning. Steps 2.a - 2.d iterate over all candidate references v_i and compute the gain obtained by replacing an existing reference with v_i. This is done as follows: Step 2.a computes the total number of sequences pruned using existing reference set for Q. Step 2.b initializes the array P. Each entry $P[a]$ of P denotes the number of sequences pruned after replacing the ath existing reference with v_i. Step 2.c iterates over all existing references. At each iteration, it replaces an existing reference with v_i and computes the total number of sequences pruned for all queries in Q. Step 2.d then computes the largest possible gain obtained by subtracting the number of sequences pruned using the original reference set from that of the best possible new reference set. If there is no gain, then the algorithm terminates and returns the current reference set (Step 3). Otherwise, it updates the reference set using the new reference that gives the best possible gain (Steps 4 to 6).

```
/*Input:Sequence database $S$,
       Number of references $m$,
       Sample query set $Q$,
  Output:Set of references $V = \{v_1, v_2, \ldots, v_m\}$*/

• Initialize $V$ with top $m$ references obtained using the Maximum Variance method.
• $S = S$ - $V$.
• Do
    1. $G[i]$ =0, $1 \leq i \leq |S|$. /* Sum of gains for each $v_i \in S$ */
    2. For each $[v_i, s_j]$, $\forall v_i \in S$, $\forall s_j \in S$ do
        (a) Let $g_o$ be the number of queries for which $s_j$ is pruned using reference in $V$.
        (b) $P[a] = 0$, $1 \leq a \leq m$. /* Initialize gain for ith reference $V$*/
        (c) For each $[e, q]$ pair, $\forall e \in V$ and $\forall q \in Q$ do
            i. $V' = V$ - $\{e\} \cup \{v_i\}$.
            ii. If(PRUNE($V'$, $q$, $s_j$) = 1) $P[e]$++.
        (d) If (MAX($P$) > $g_o$ ) do
            − $G[i]$+ = MAX($P$) - $g_o$.
    3. If (MAX($G$) $\leq 0$) Return $V$.
    4. Let $v = argmax_i(G[i])$.
    5. Update $V$ with $v$.
    6. $S = S - \{v\}$.
  While (true).
```

Figure 5: Maximum Pruning algorithm: PRUNE(V', q, s) returns true if one of the references in V' can prune s. MAX(P) (MAX(G)) returns the maximum of the values in P (G).

Computational Complexity: The algorithm needs access to distances between all pairs of sequences in S (Step 2). This requires $O(N^2)$ sequence comparisons. Note that this a one time cost, and is not incurred at each iteration of the Do-While loop. The PRUNE function requires the distances between all (query sequence, database sequence) pairs. We pre-compute these distances once. This requires $N|Q|$ sequence comparisons. So, the total number of sequence comparisons is $O(N^2 + N|Q|)$. Step 2 has to consider $O(N^2)$ pairs of sequences. For each pair it computes the gain for all queries in Q after replacing sequences in the reference set one by one. This takes $O(m|Q|)$ time since a new reference can replace any of the m references (Step 2.c). Thus overall time taken by this algorithm is $O(N^2m|Q|)$ for each iteration of the Do-While loop. There can be at most N iterations, leading to the worst case complexity $O(N^3m|Q|)$.

5.2 Sampling-Based Optimization

Although the Maximum Pruning algorithm in Section 5.1 is faster than a purely combinatorial approach, it is still impractical for large databases. To address this problem, we propose two sampling-based optimizations to improve the complexity of the algorithm by reducing the number of (sequence, reference) pairs processed. The first one reduces the number of sequences and the second one reduces the number of references.

Estimation of gain: Our first optimization reduces the number of sequences that must be considered when computing the gain associated with a new reference point. This algorithm replaces Step 2 of the Maximum Pruning algorithm in Figure 5. We estimate gain based on a small sample of the database rather than the entire database. One of the most important technical considerations in the design of this algorithm is how to decide whether the gain estimate is accurate enough based upon the sample.

```
/*Input:Sequence database $S$,|$S$| = $N$.
        Sample queries $Q$,|$Q$| = $q$.
  Output:$G[i]$ $\forall v_i \in S$.
 */
For each $v_i \in S$ do
  1. $\hat{g_1} = \hat{g_2} = 0$. /* Initialize total and estimate for 2nd moment */
  2. $\alpha = 0$. /* Initialize sampling fraction */
  3. Do
      (a) Select a random $s_j \in S$, where $s_j \neq v_i$.
      (b) $newGain$ = GAIN($v_i, s_j, Q$).
      (c) $\hat{g_1}$ += $newGain$; $\hat{g_2}$ += $newGain^2$.
      (d) $\alpha = \alpha + 1/|S|$.
      (e) $\hat{\sigma^2} = \hat{g_2}/\alpha^2 - \hat{g_1}^2/(|S|\alpha^3)$.
     While $2\alpha\hat{\sigma}/\hat{g_1} > \epsilon$.
  4. $G[i] = \hat{g_1}/\alpha$.
```

Figure 6: Optimization 1: Estimation of gain.

Figure 6 presents our sampling algorithm in detail. The algorithm takes a database of sequences S, query set Q and an error rate ϵ as input. It returns the total gain obtained by replacing an existing reference with a new reference for all possible new references. For each candidate reference v_i, Step 3.a iteratively selects a random sequence $s_j \in S$. Step 3.b then computes the gain by using v_i as a reference for s_j for Q. The gain is computed as follows. Steps 2.a to 2.c of the Maximum Pruning algorithm in Figure 5 are executed to compute the total pruning achieved with respect to s_j by replacing each existing reference with v_i. Then the gain is given by the best pruning over all possible replacements. Step 3.c updates the total gain seen as well as the total squared gain seen (which can be used to estimate the second moment of the gains that have been sampled). Step 3.d updates the sampling fraction, and Step 3.e then computes an estimate for the variance of our gain estimate. The algorithm terminates when the desired accuracy is reached. The accuracy of the gain estimate is guessed at by making use of the Central Limit Theorem (CLT), which implies that errors over estimated sums and averages of this type are normally distributed. Since 95% of the standard normal distribution's mass is within two standard deviations of zero, if we treat the current gain estimate $\hat{g_1}/\alpha$ as the true gain and terminate when twice the relative standard deviation is less than ϵ, we can be assured that the relative error is less than ϵ with 95% probability.

For an average sample size of f with $f \ll N$, this approach reduces the complexity to $O(N^2fm|Q|)$. This is because we iterate over f sequences rather than all N sequences while computing gain.

Estimation of largest gain: The Maximum Pruning algorithm in Figure 5 uses all database sequences as candidate references to select the reference set. Our second optimization reduces the number candidate references processed in each iteration with the help of sampling. The goal is to use a small subset of the database as candidate references, and yet achieve pruning rates close to Maximum Pruning.

Formally, we define the problem as follows. Let $G[i]$ be the gain that can be achieved by indexing with the ith reference. Assume that $e = argmax_i(G[i])$ (i.e., $G[e]$ has the largest gain). Given two parameters τ and ψ, where $0 \leq \tau,\ \psi \leq 1$, we want to sample the candidate reference set ensuring that the largest gain from this sample is at least $\tau G[e]$ with probability ψ.

Since $G[e]$ is not known in advance, we can use the Type-I Extreme Value Distribution (also known as the Gumbel distribution [14]) to estimate its value. This is done as follows. We assume that each $G[i]$ is produced by sampling repeatedly from a normally distributed random variance. The first step is to determine the mean and standard deviation of this variable. To do this, we sample a set of candidate references and compute the mean μ and the standard deviation σ of the gains for the sample. These are taken as the mean and standard deviation of the underlying distribution.

Since the values in G are assumed to be samples from a normal distribution, the largest gain $G[e]$ is known to have a Gumbel distribution whose parameters can be computed using μ and σ. Let N and t be the number of candidate references and the sample size, respectively. The two parameters of the Gumbel distribution, referred to as location parameter a and a scale parameter b, are computed as follows:

$$a = \sqrt{2 \log N}$$

$$b = \sqrt{2 \log N} - \frac{\log \log N + \log 4\pi}{2\sqrt{2 \log N}}$$

The mean of the corresponding Gumbel distribution is then calculated as:

$$\hat{\mu} = \sigma[\frac{-\log_e(-\log_e(0.5))}{a} + b] + \mu$$

This tells us exactly what the expected value of the gain associated with the best reference sequence is.

Thus, we can stop sampling once the best gain (with high probability) is *almost* good enough; that is, we stop when it is at least $\tau\hat{\mu}$. To compute how many samples are needed, let $P(x < \tau\hat{\mu})$ be the probability that the gain of a random reference x is less than $\tau\hat{\mu}$. This probability can be calculated from the distribution of $G[i]$. The probability that the gain of all the t randomly selected references are less than $\tau\hat{\mu}$ is $P(x < \tau\hat{\mu})^t$. Solving the inequality:

$$1 - P(x < \tau\hat{\mu})^t \geq \psi$$

gives the number of samples needed (denoted by t) as:

$$t \leq \frac{\log_e(1-\psi)}{\log_e(P(x < \tau\hat{\mu}))}.$$

In our experiments we have used $\tau = \psi = 0.99$. Each iteration of the Do-While loop of the Maximum Pruning algorithm in Figure 5 computes the gain from random candidate references until the required accuracy is reached. Each iteration uses a different sample of candidate references. If the average sample size of the m iterations is h, $h \ll N$, then this optimization along with the first optimization reduces the overall complexity to $O(Nfhm|Q|)$.

6. ASSIGNMENT OF REFERENCES

So far, we have discussed how to select reference sequences. In this section, we discuss how to use them. Traditional reference-based indexing methods such as Omni, use the same references to index all the database sequences. That is, given k references, Omni tries to prune all the database sequences using the same k references.

We develop a new strategy to assign references to database sequences. We increase the total number of references but use only a subset of them to index each database sequence. Formally, given a set of m references ($m > k$), our goal is to assign a set of k references to each database sequence such that at least one of these k references will remove s from the candidate set for as many queries as possible. As we will discuss later in more detail, when m is much smaller than the database size, this strategy improves the performance with little increase in the storage cost.

The set of m initial references (V) can be chosen in many different ways. We use one of our two selection strategies given in Sections 4 and 5. Let Q be a sample query set. For each database sequence s, we greedily assign the k references in V that provide the best pruning according to Q. This approach is similar to reference selection strategy of Maximum Pruning in Section 5. There are two main differences. First, the k references are selected for each database sequence independently. Second, the total number of references can be greater than k. This potentially increases the cost of query-reference comparisons. Thus, a new reference is assigned to a sequence only if the benefit of including this reference is more than the additional cost it brings.

Figure 7 shows the algorithm for assigning references to each database sequence. The algorithm takes a query set Q, database S, potential reference set V, and the number of references per sequence k as input. It returns a mapping from each database sequence to k references in V. For each database sequence s, the algorithm iteratively maps one reference until k references are mapped as follows. It maintains an array $Vcount$, where $Vcount[i]$ gives the number of queries for which a sequence is pruned using the reference $v_i \in V$. The algorithm selects the reference $e \in V$ for which s is pruned for maximum number of queries (Steps 3.b-3.c). Steps 3.e and 3.f removes e from V and map it to s. Step 3.g removes the from the query set those queries for which s was pruned using e. This is done to ensure that the new references are selected only to improve the queries for which s can not be pruned using the existing references. Each reference costs $|Q|$ extra sequence comparisons. This is because each query needs to be compared with all the references. Thus, a reference is useful only if it prunes a total of more than $|Q|$ sequences for all the queries in Q. If the total gain from a reference is not greater than $|Q|$, it is removed from the reference set (Step 4). The reference sets of sequences which have less than k references are then updated with new references from V (Step 5).

Computational Complexity: The algorithm needs access to distances between all (query sequence, reference sequence) pairs. This takes O($tm|Q|$), where t is the time taken for one sequence comparison. If the selection strategy is Maximum Pruning, then the pre-computed query-reference distances are used. In each of the k iterations, Step 3.b of the algorithm can process up to O($m|Q|$) query-reference pairs. Thus the overall time taken by the algorithm

```
/*Input:Sequence database $S$,|$S$| = $N$.
        Reference set $V$,|$V$| = $m$.
        Sample queries $Q$,|$Q$| = $q$.
        References per sequence $k$.
  Output:$E = \{E_1, E_2, \ldots, E_N\}$,
        $E_s$ is assigned to sequence $s \in S$.
*/
  1. $G[i] = 0, 1 \leq i \leq m$. /* Total gain from each reference
     $v_i \in V$ */
  2. $E_i = \{\}, 1 \leq i \leq N$. /*Initialize reference set of each
     sequence*/
  3. For each $s \in S$ do
       • Repeat
         (a) $Vcount[i] = 0, 1 \leq i \leq m$. /* Initialize gain
             for $[v_i, s]$ pair */
         (b) For all $[v, Q_j]$ , $\forall v \in V$ and $\forall Q_j \in Q$ do
               – If(PRUNE($s, Q_j, v$)) do $Vcount[v]$++.
         (c) Let $e = argmax_x(Vcount[x])$.
         (d) $G[e]$+= $Vcount[e]$.
         (e) $V = V - \{e\}$.
         (f) $E_s = E_s \cup \{e\}$.
         (g) Remove from $Q$ queries for which $s$ is pruned
             with reference $e$.
         Until $|E_s| = k$.
       • Re-insert all deleted entries from sets $V$ and $Q$.
  4. For all $v \in V$ do
       • If($G[v] \leq |Q|$) $V = V - \{v\}$.
  5. Update the reference sets $E_s$ $\forall s \in S$.
```

Figure 7: Algorithm to assign references to sequences: The PRUNE(s, q, v) function returns true if the reference v can prune s for the query q.

is O($Nmk|Q|$).

7. SEARCH ALGORITHM

We have discussed how to find references (Sections 4 and 5) and how to map them to database sequences (Section 6). In this section, we will discuss how we use the mapped references to answer range queries.

Let S be the database of N sequences and V be the reference set. The set of k references mapped to s_i is given by E_i, $\forall s_i \in S$. Here, $E_i \subseteq V$. We pre-compute the edit distances $ED(v, s_i)$, $\forall s_i \in S$ and $\forall v \in E_i$. This is a one time cost for the database. For a query $q \in Q$ and range ϵ, we first compute the edit distances from q to the reference sequences, i.e. $ED(q, v_i)$ $\forall v_i \in V$. For each (q, s_i) we compute the lower and upper bounds LB and UB as given in Section 2. Depending on LB, UB, and ϵ, we insert s_i into one of the two sets *Result set* and *Candidate set* as follows. If $UB \leq \epsilon$, we insert s_i into *Result set*. If $LB \leq \epsilon \leq UB$, we insert s_i into *Candidate set*. Otherwise, we prune s_i. Once the *Candidate set* is determined, we do the actual sequence comparison between q and all the sequences in the *Candidate set* to filter out false positives.

Computational Complexity: Given a sequence database S with N sequences, the selection strategy selects m references. The assignment strategy maps each sequence with k, $k \leq m$, references. For each sequence s and its reference $v_i \in V$, its mapping is of the form $[i, ED(s, v_i)]$. The Nk edit distances between the sequences and references mapped to the sequences are stored in a file. The selection strategy, mapping database sequences $s \in S$ to references, and computing the $[s, v_i]$ edit distances, are all one-time costs for a sequence database.

During database search, references and reference-to-sequence edit distances are loaded into the memory. With an average sequence size of z bytes and four bytes for an integer, this requires $(8Nk + mz)$ bytes of memory (Section 2). Here eight bytes are used to store each of the Nk mappings. With an increase in m, the sequences can be assigned better references. However, this will increase the number of query-to-reference computations. Hence, in our experiments we restrict m to a fraction of the database size.

Given a query set Q, the edit distances of all (query, reference) pairs are computed. This involves $m|Q|$ sequence comparisons. For every (query, sequence) pair, all k references for the sequence are compared to compute the lower and upper bounds. This takes O($Nk|Q|$) time. If C_m is the average candidate set size for Q using the m references, it takes $C_m|Q|$ sequence comparisons to get the final result. For an average sequence length of L, the sequence comparison takes O(L^2) time. Thus, the overall time taken by the search algorithm is O($(m + C_m)|Q|L^2 + Nk|Q|$).

Table 1: A list of our methods: Each of the two selection methods has two types of assignments.

Assignment	Selection Strategies	
Strategy	Max. Variance	Max. Pruning
Same References	MV-S	MP-S
Diff. References	MV-D	MP-D

8. EXPERIMENTAL EVALUATION

In this section, we compare the performance of different methods for sequence indexing based on the number of sequence comparisons performed.

Table 1 lists the methods proposed in this paper that we compare. For MV-D and MP-D, we have selected 200 references from our two selection strategies (i.e. $m = 200$) unless otherwise stated. For MV-S and MP-S, we have selected the top k ($k = m$) references. We compare our methods with several other existing methods: 1) Omni [11] 2) frequency vectors [19] (FV), 3) the M-Tree [8], 4) the DBM-Tree [33], 5) the Slim-Tree [30] and 6) the DF-Tree [29]. The C++ implementations of the M-Tree, the DBM-Tree, the Slim-Tree and the DF-Tree were obtained from the Arboretum MAM library (`http://gbdi.icmc.usp.br/arboretum/`). We have used the following three types of databases in our experiments:

Text database: We created text database from 33 randomly selected books from the Gutenberg project (`http://www.gutenberg.org/`). The alphabet consists of 36 alphanumeric characters. The database contains 8,000 sequences of length 100.

DNA Database: We use the *Escherichia Coli* or *E.Coli* (K-12 MG1655) genome from GenBank [5] (`ftp://ftp.ncbi.nih.gov/genbank/`). The alphabet size of a DNA sequence is 4 (A,C,G,T). We created four databases of non-overlapping sequences of lengths 25, 50, 100 and 200. Each database is obtained by chopping the E.Coli database into 20,000 non-overlapping sequences. Databases of different sizes containing 5,000, 10,000 and 15,000 non-overlapping sequences, each of length 100, are also created.

Protein database: Protein sequences in the Eukaryota kingdom of organisms are used for this database. They are obtained from SwissProt [4] (`ftp://ftp.ebi.ac.uk/pub/databases/swissprot/`). The alphabet size is 20. A set of 4,000 sequences having up to 500 amino acids is selected randomly.

For each database, 200 sequences from different parts of the same species/database are selected. 100 sequences are used as sample queries for Maximum Pruning and another 100 sequences are used as query sequences. Note that the query sequences and database sequences for the DNA databases are taken from different parts of the same species and they do not overlap. Similarly for the protein data, the query sequences are taken from proteins from a different part of the database without any overlap. For the genome data, the mean distance between the query and the database sequences is 56 and for the protein data, the mean distance is 192. We have also created three additional query sets that we test in Section 8.2.3: one from each of the organisms *Danio Rerio* (Zebrafish), *Mus Musculus* (Mouse) and *Heliconius Melpomene* (Butterfly). Each query set contains 100 sequences of length 100 and each is selected randomly from its organism.

We ran our experiments on an Intel Pentium 4 processor running Linux with 2.8 GHz clock speed and 1GB of main memory.

8.1 Comparison of our methods

In this section, we present experimental results comparing our methods under different parameter settings.

8.1.1 *Impact of Query Range (ϵ)*

The goal of this experiment set is to understand the behavior of our methods for different query ranges. Here, we compare the performance of MV-S, MP-S, MV-D and MP-D with $\epsilon = 2$ to 32 for the DNA sequences and $\epsilon = 60$ to 420 for the protein database. The number of references used for the DNA and protein database is 4 and 32, respectively. The plots are given in log-scale.

Figure 8(a) presents the number of sequence comparisons for the DNA database. This increases with the query range for all four methods. For different ranges, MP-D and MV-D have a smaller number of sequence comparisons compared to MV-S and MP-S. MP-D is gives the best results. For ranges up to 8, assigning different reference sets to each sequence results in a significant reduction of sequence comparisons for both selection strategies. This shows that assigning different reference sets to each sequence gives better pruning results than the traditional approach of assigning the same references to all sequences. Using either a uniform or varying assignment of sequences to references, MP performs slightly better than MV. This is due to the fact that MP is using knowledge of the input query distribution. Figure 8(b) presents the results for the protein database. MP-D outperforms the other methods for all query ranges. For most of the query ranges, the MP methods perform better than the MV methods. We have observed similar results for the text database.

8.1.2 *Impact of Number of References (k)*

The goal of this experiment is to understand the behavior of our methods for different numbers of references. Here, we compare MV-S, MP-S, MV-D and MP-D by fixing the query range and varying the number of references assigned, $k = 2, 4, 8, 16$ and 32. We use both the protein and DNA databases. The plots are in log-scale for the DNA database.

Figure 9(a) shows the number of sequence comparisons for the DNA database. For all four methods, the number of sequence comparisons decreases dramatically with an increase in k. As the number of references increases from 2 to 32, the number of sequences compared drops by factors of 5 to 20 for the methods MP-D and MP-S. Figure 9(b) presents the results for the protein database. With an increase in the number of references, there is a gradual decrease in the number of sequence comparisons. The MV-S strategy outperforms the MP-S strategy at $k = 8, 16$ and 32 for the protein database and MP-S outperforms MV-S for all values of k in the DNA sequence database. The experiments using the text database gave similar results to the DNA sequence database, with MP-D giving the best results. This shows that with an increase in the number of references, memory can be better utilized by assigning a subset of references to each database sequence.

8.1.3 *Impact of m*

The goal of this experiment set is to understand the behavior of MP-D for different cardinalities of the reference set, $|V| = m$. The number of references is 8 and the query range is 8. We use the DNA sequence database. The plots are given in log-scale. Figure 10 gives the number of sequence comparisons for different reference set cardinalities. Up to $m = 200$, the number of sequence comparisons reduces at a fast rate. From $m = 200$ to 500, there is very little improvement in performance. For $m > 500$, the number of sequence comparisons increases. Hence, in our subsequent experiments we use $m = 200$.

8.2 Comparison with existing Methods

In this section we compare MV-D and MP-D with Omni, FV, the M-Tree [8], the DBM-Tree [33], the Slim-Tree [30] and the DF-Tree [29].

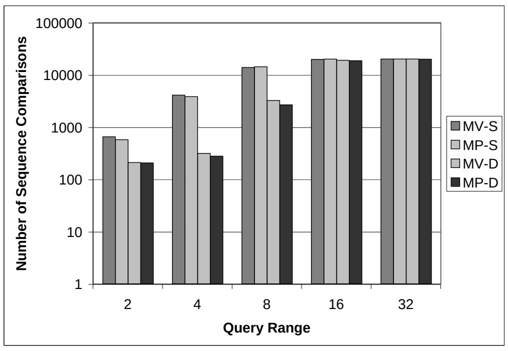
8.2.1 *Index Construction Time*

The time taken to construct each index is given as the first line in Table 2. I_C denotes the index construction time. For the tree-based methods, I_C refers to the time taken to construct the tree structure. For Omni, I_C is the time taken to generate 16 references. I_C for MP-D is the time taken to generate 200 references using the maximum pruning method with sampling-based optimization. This includes the time taken for selecting the references and the index construction time, where the reference-to-data sequence distances are recorded.

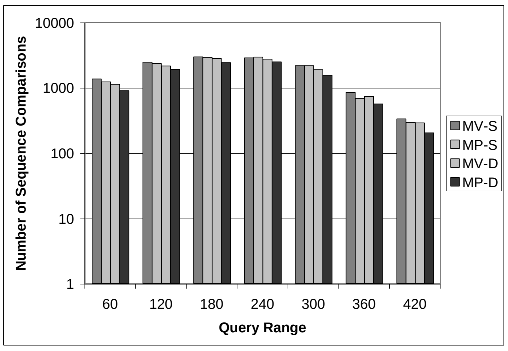
8.2.2 *Impact of Query Range (ϵ)*

The goal of this experiment set is to compare the behavior of our methods with several existing methods for different query ranges. For the protein database, the query range varied from $\epsilon = 60$ to 420 with 32 references. We test values of ϵ from 2 to 32 for the DNA database and from $\epsilon = 15$ to 75 for the text database. The plots are given in log-scale.

Table 2 presents results for the six existing methods along

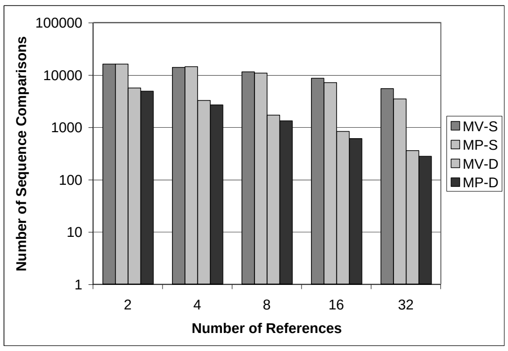


(a) DNA Sequence dataset

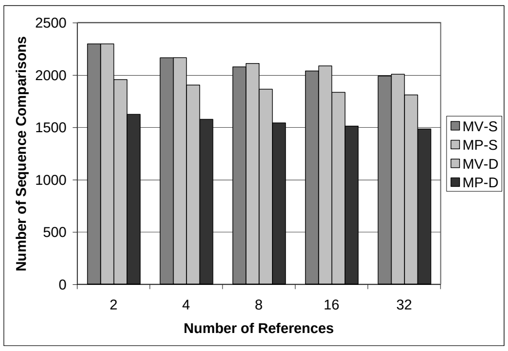


(b) Protein dataset

Figure 8: Number of sequence comparisons for MV-S, MP-S, MV-D and MP-D for the DNA and protein databases for queries with varying ranges. The number of references is 4 for DNA and 32 for the protein database.



(a) DNA sequence dataset



(b) Protein dataset

Figure 9: Number of sequence comparisons for MV-S, MP-S, MV-D and MP-D with the DNA sequence and protein databases for a varying number of references. Here query range = 8 for DNA database and 300 for protein database.

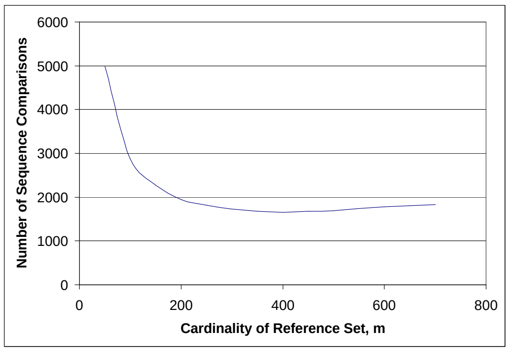


Figure 10: Impact of m: Number of comparisons of MP-D for DNA sequence database for values of m. The number of references is 8 and query range is 8.

with MV-D and MP-D for the DNA sequence database. With increasing in query range, the number of sequence comparisons increases for all the methods. The tree-based methods compare more sequences than a simple sequential scan with a large query range. This is due to the comparison of the query sequence with the sequences in the intermediate tree nodes. For ranges 8 to 32, Omni performs more sequence comparisons than FV, MV-D and MP-D. As we increase the range from 2 to 8, MP-D reduces the number of comparisons by a factor of up to 6 to 100. For larger ranges, MP-D reduces the number of sequence comparisons up to a factor of 2. Thus, MP-D generally outperforms all other methods.

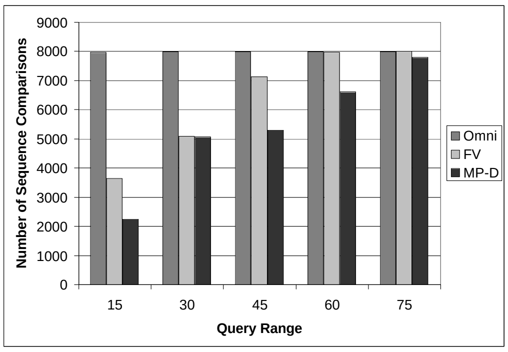
In the remaining experiments, we only use Omni and FV for comparison to our methods as they perform the best among all competitors. Figure 11(a) presents the results for the text database. MP-D outperforms the other methods for all query ranges. For example, at $\epsilon = 15$ to 75, MP-D compares up to 3 times fewer sequences than Omni, and up to 50% fewer sequences than FV. Figure 11(b) gives the results for range queries on the protein database. FV outperforms its competitors at ranges of 60 and 120. As the query range increases, MP-D outperforms FV.

8.2.3 Impact of Number of References (k)

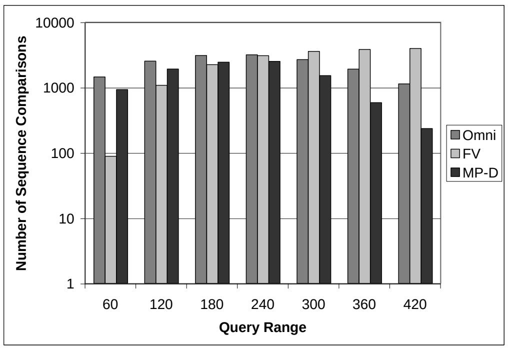
The goal of this experiment set is to compare the behavior of our methods with existing methods using different numbers of references. Here, we compare the performance of Omni, FV and MP-D by fixing the query range for a varying number of references $k = 2, 4, 8, 16$ and 32. We use the

Table 2: Number of sequence comparisons of existing methods for the DNA sequence database with varying query ranges. I_C denotes the index construction Time. QR denotes the query range. ss and ms denote the running time in seconds and minutes respectively. Number of references for Omni, DF-Tree, MV-D and MP-D is 16.

	M-Tree	Slim-Tree	DBM-Tree	DF-Tree	Omni	FV	MV-D	MP-D
QR	I_C=50 ms	I_C=15 ms	I_C=14 ms	I_C=480 ms	I_C=14 ms	I_C=6 ss	I_C=74 ms	I_C=180 ms
2	9946	10228	7313	8041	228	247	205	200
4	13426	14390	10963	10773	1202	1264	273	208
8	20147	19691	17507	19301	12677	6208	2648	1126
16	22865	21176	19977	24861	19927	16088	18541	18296
32	22892	21192	19997	25021	20000	19811	19840	19836



(a) Text dataset



(b) Protein dataset

Figure 11: Number of sequence comparisons for Omni, FV and MP-D on text and protein databases for queries with varying ranges. Here the number of references is 8 for text database and 32 for protein database.

DNA and protein databases. The query range is 8 for the DNA database and 300 for the protein database. Since FV does not use references, we have compared the results of the other methods with different numbers of references to a single frequency vector. The plots are given in log-scale.

Figure 12(a) shows the number of sequence comparisons for all three methods for the DNA database. The results show that even with two references per sequence, MP-D outperforms the other methods. As the number of references increases, the number of sequence comparisons used by MP-D is smaller by up to a factor of 25 compared to Omni and up to a factor of 43 compared to FV. Figure 12(b) gives the results for the protein database. For a varying number of references, the reference-based methods perform better than FV. MP-D reduces the number of comparisons by a factor of 2.7 compared to FV and a factor of 2 compared to Omni.

8.2.4 *Impact of Input Queries*

In this experiment, we evaluate our methods when the distribution of queries differs from that of the sample query sequences used in reference selection. We used three query sets from three different species (Mouse, Zebrafish and Butterfly) which are taxonomically distant from the species we used in our database (E.coli). We used 8 references for Omni and MP-D.

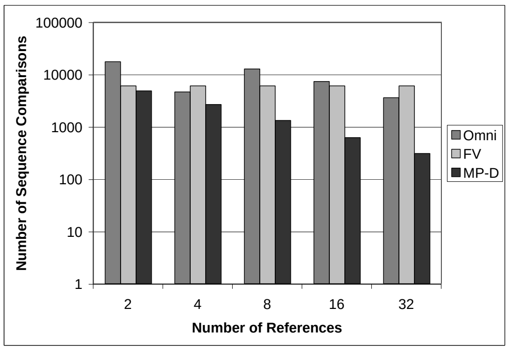
Table 3 gives the results. In comparison to Table 2, for most of the query ranges, MP-D reduces the number of sequence comparisons by up to a factor of 10 compared to FV and by up to a factor of 20 compared to Omni. With an increase in query range, the number of sequence comparisons for all query sets increases. The performance of MP-D is not affected by a change in query distribution. At a query range of 16, FV performs slightly better than MP-D for all query distributions. Omni requires more sequence comparisons than FV and MP-D for all query ranges.

8.2.5 *Scalability in Database Size*

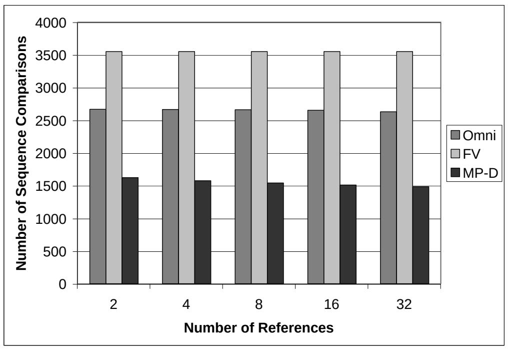
Next, we test the Omni, FV and MP-D methods for scalability in database size. We have used four DNA sequence databases with 5,000, 10,000, 15,000 and 20,000 sequences each. The number of references used by Omni and MP-D is 32 and the query range is 8. Figure 13 gives the results. With an increase in the size of the database, MP-D outperforms all other methods, and FV uses more sequence comparisons than Omni. The rate of increase in comparisons is slowest for MP-D.

8.2.6 *Scalability in Sequence Length*

The goal of this experiment set is to compare the behavior of our method with existing methods for increasing lengths of sequences. Here, we compare the Omni, FV and MP-D methods. We have used four DNA sequence database having 10,000 sequences and with sequence lengths of 25, 50, 100 and 200. The number of references used in the methods Omni and MP-D is 32, and the query range is 8. Figure 14 gives the results. All of the methods show reduction in the number of sequence comparisons with an increase in the sequence lengths. For shorter sequences, a range of 8 is large relative to the sequence length. In these cases MP-D outperforms Omni and FV by a factor of 2. For longer sequences, a range of 8 is relatively small. For these sequences, MP-D reduces the number of sequence comparisons by a factor



(a) DNA sequence dataset



(b) Protein dataset

Figure 12: Number of sequence comparisons Omni, FV and MP-D over the DNA sequence and protein databases for a varying number of references. Here query range = 8 for DNA database and 300 for protein database.

Table 3: Number of sequence comparisons over the DNA sequence database for different query sets with varying query ranges. QR gives the query range. HM, MM and DR are the three query sets from DNA sequences of three different organisms *Heliconius Melpomene*, *Mus Musculus* and *Danio Rerio* sequence respectively. The number of references for Omni and MP-D is 8.

	HM			MM			DR		
QR	Omni	FV	MP-D	Omni	FV	MP-D	Omni	FV	MP-D
2	428	321	249	716	457	222	592	424	244
4	1217	1360	401	1225	1385	408	1224	1381	419
8	6489	1847	1656	9514	2718	1259	7433	2279	1444
16	18807	8296	14526	19631	11644	16550	19236	9300	15408
32	20000	19045	18558	20000	19418	19146	20000	19043	18945

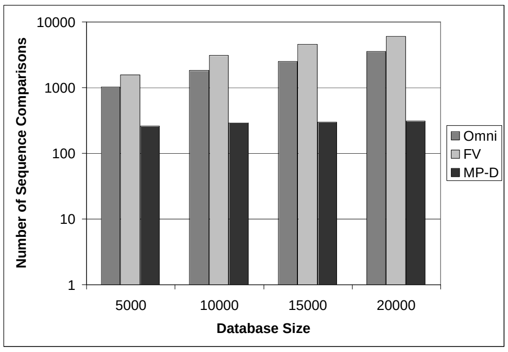


Figure 13: Scalability in database size: Number of sequence comparisons for Omni, FV and MP-D for the DNA sequence database for varying database sizes. The number of references is 32 and query range is 8.

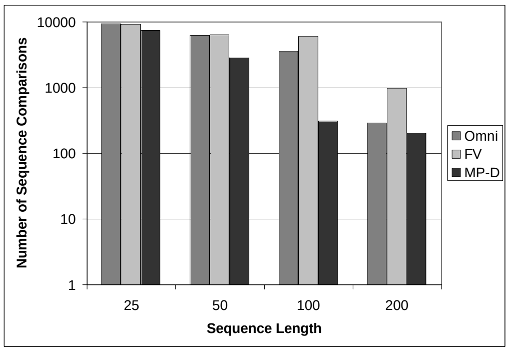


Figure 14: Scalability in sequence length: The number of sequence comparisons for Omni, FV and MP-D for DNA database for varying sequence lengths. The number of references is 32 and the query range is 8.

of 20 compared to FV and Omni. As the sequence length increases, Omni outperforms FV.

9. DISCUSSION

In this paper, we considered the problem of similarity search in a large sequence databases with edit distance as the similarity measure. We developed a family of reference-based indexing techniques. We developed two novel methods for selecting reference sequences. Unlike existing methods, our methods select references that represent all parts of the database. Our first method, *Maximum Variance*, maximizes the spread of database around the references. Our second method, *Maximum Pruning*, optimizes pruning based on a set of sample queries. We also developed sampling methods to improve the running time of the index construction. We propose mapping different references to each database sequence dynamically rather than using the same references. We developed a sampling strategy which finds a mapping of references to sequences that maximizes the pruning rate with a given probability.

According to our experiments, our methods perform much better than existing strategies including Omni and frequency vectors. Among our methods, Maximum Pruning with dynamic assignment of reference sequences performs the best. The total cost (number of sequence comparisons) of our methods are up to 20 and 30 times less than that of Omni and frequency vectors, respectively.

10. REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] R. Baeza-Yates and C. Perleberg. Fast and practical approximate string matching. In *CPM*, pages 185–192, 1992.

[3] R. A. Baeza-Yates and G. Navarro. Faster Approximate String Matching. *Algorithmica*, 23(2):127–158, 1999.

[4] A. Bairoch, B. Boeckmann, S. Ferro, and E. Gasteiger. Swiss-Prot: juggling between evolution and stability. *Briefings in Bioinformatics*, 1:39–55, 2004.

[5] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, B. Rapp, and D. Wheeler. GenBank. *Nucleic Acids Research*, 28(1):15–18, January 2000.

[6] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *ACM SIGMOD*, pages 357–368, 1997.

[7] X. Cao, B. C. Ooi, H. Pang, K.-L. Tan, and A. K. H. Tung. Dsim: A distance-based indexing method for genomic sequences. In *BIBE*, pages 97–104, 2005.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *The VLDB Journal*, pages 426–435, 1997.

[9] A. Delcher, S. Kasif, R. Fleischmann, J. Peterson, O. Whited, and D. Salzberg. Alignment of Whole Genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[10] P. Ferragina and R. Grossi. The String B-tree: A New Data Structure for String Search in External Memory and Its Applications. *JACM*, 46(2):236–280, 1999.

[11] R. F. S. Filho, A. J. M. Traina, C. Traina, and C. Faloutsos. Similarity Search without Tears: The OMNI Family of All-purpose Access Methods. In *ICDE*, pages 623–630, 2001.

[12] E. Giladi, M. Walker, J. Wang, and W. Volkmuth. SST: An Algorithm for Finding Near-Exact Sequence Matches in Time Proportional to the Logarithm of the Database Size. *Bioinformatics*, 18(6):873–877, 2002.

[13] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[14] E. J. Gumbel. *Statistics of Extremes*. Columbia University Press, New York, NY, USA, 1958.

[15] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1 edition, January 1997.

[16] E. Hunt, M. P. Atkinson, and R. W. Irving. A Database Index to Large Biological Sequences. In *VLDB*, pages 139–148, Rome, Italy, September 2001.

[17] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005.

[18] T. Kahveci, V. Ljosa, and A. Singh. Speeding up whole-genome alignment by indexing frequency vectors. *Bioinformatics*, 2004. to appear.

[19] T. Kahveci and A. Singh. An Efficient Index Structure for String Databases. In *VLDB*, pages 351–360, Rome,Italy, September 2001.

[20] J. Kärkkäinen. Suffix Cactus: A Cross between Suffix Tree and Suffix Array. In *CPM*, 1995.

[21] U. Manber and E. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[22] E. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *JACM*, 23(2):262–272, 1976.

[23] C. Meek, J. M. Patel, and S. Kasetty. OASIS: An Online and Accurate Technique for Local-alignment Searches on Biological Sequences. In *VLDB*, 2003.

[24] E. W. Myers. An o(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[25] G. Navarro and R. Baeza-Yates. A Hybrid Indexing Method for Approximate String Matching. *J. Discret. Algorithms*, 1(1):205–239, 2000.

[26] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *JMB*, 48:443–53, 1970.

[27] W. Pearson and D. Lipman. Improved Tools for Biological Sequence Comparison. *PNAS*, 85:2444–2448, April 1988.

[28] S. Sahinalp, M. Taşan, J. Macker, and Z. Özsoyoğlu. Distance Based Indexing for String Proximity Search. In *ICDE*, 2003.

[29] C. Traina, A. J. M. Traina, R. F. S. Filho, and C. Faloutsos. How to improve the pruning ability of dynamic metric access methods. In *CIKM*, pages 219–226, 2002.

[30] C. Traina, A. J. M. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In *EDBT*, pages 51–65, 2000.

[31] E. Ukkonen. Algorithms for Approximate String Matching. *Information and Control*, 64:100–118, 1985.

[32] E. Ukkonen. On-line Construction of Suffix-trees. *Algorithmica*, 14:249–260, 1995.

[33] M. R. Vieira, C. Traina, F. J. T. Chino, and A. J. M. Traina. DBM-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data. In *SBBD*, pages 163–177, 2004.

[34] J. Vleugels and R. Veltkamp. Efficient image retrieval through vantage objects. In *VISUAL*, pages 575–584. Springer, 1999.

[35] P. Weiner. Linear Pattern Matching Algorithms. In *IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[36] P. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *SODA*, pages 311–321, 1993.